# Facultad de Informática
# Grado en Ingeniería Informática

## Lógica

### PARTE 4: RESOLUCIÓN

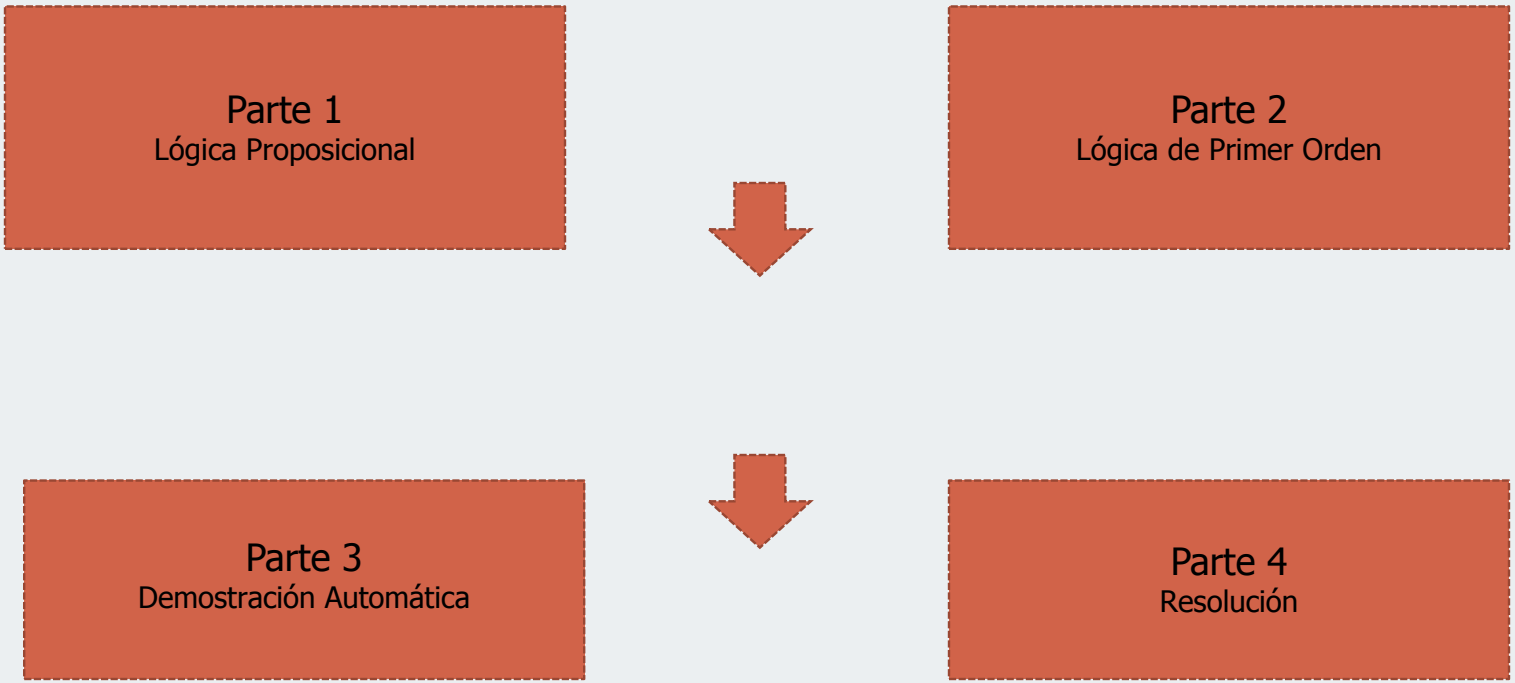
# Tema 16: Extracción de respuestas e introducción a la programación lógica

Profesor: Javier Bajo
jbajo@fi.upm.es

# Introducción.



## ❑ Componentes



# Resolución SLD.



- El procedimiento de demostración automática estudiado se basa en:
  - un formalismo de representación de conocimiento adecuado: **forma clausular**
  - un cálculo deductivo correcto y completo: **resolución con unificación**
- La eficiencia del cálculo puede mejorarse utilizando estrategias: **lineal, input, dirigida y ordenada**
- Pero, en principio, no todas las estrategias garantizan que si la deducción es correcta se pueda demostrar (**no todas son completas**)
- **¿Habría alguna manera de poder aplicar todas las estrategias, mejorando al máximo la eficiencia del cálculo, sin perder la completitud?**

# Resolución SLD.



- **Resolución SLD.** Estrategia que combina las estrategias lineal, input, dirigida y ordenada sobre un tipo particular de cláusulas: definidas o de Horn.

- **Cláusula de Horn:** cláusula que tiene como máximo un literal afirmado

  A v ¬B1 v ¬B2 (*el literal afirmado, cuando existe, se coloca al inicio de la cláusula*)

  A

  ¬B1 v ¬B2

- Aquellas cláusulas que no tienen literal afirmado forman parte del **conjunto objetivo**.

- Aquellas cláusulas que tienen un literal afirmado forman parte del **conjunto soporte**.

- La **resolución SLD es completa**: Un conjunto de cláusulas de Horn es insatisfacible sii existe una refutación SLD de dicho conjunto

# Extracción de respuestas.



- Por tanto, si representamos el conocimiento disponible mediante cláusulas de Horn podemos aplicar un cálculo deductivo sumamente eficiente (resolución SLD) con el que demostrar la corrección de argumentos.

- Las técnicas de demostración automática de teoremas, como la resolución, pueden aplicarse para **diseñar sistemas de extracción de respuestas y resolución de problemas** (Green, Raphael, 1968)

- La **idea** es que el conjunto de hechos necesarios para obtener una respuesta (o resolver un problema) pueden verse como axiomas de una teoría o premisas, y la pregunta (o el problema) puede verse como un teorema a demostrar o conclusión

# Extracción de respuestas.



- Tipos de preguntas (según la forma de la respuesta):
  - **Tipo A:** La clase de pregunta cuya respuesta es "si" o "no" (ej. "Sí, Luís está en Madrid" o "No, Luís no está en Madrid")
  - **Tipo B:** La clase de pregunta cuya respuesta requiere algo del tipo "dónde está", "quién es" o "bajo qué condiciones" (ej. "Juan está en Londres", "David es el marido de Bea", "Alex debería correr si llueve")
  - **Tipo C:** La clase de pregunta cuya respuesta es una secuencia de acciones (ej. "Ve a Barcelona en tren y después coge el avión a Roma")
  - **Tipo D:** La clase de pregunta cuya respuesta incluye la verificación de condiciones (ej. "Si hay plazas en el AVE, ve a Barcelona en tren, si no, ve en autobús")

# Extracción de respuestas: Tipo A.



- La respuesta es sólo "si" o "no", con lo que se puede obtener resolviendo el problema de deducibilidad asociado:

**¿Es cierto P, sabiendo A1,…,An? ➔ [A1,…,An] |- P**

- Ejemplo:

| | |
|---|---|
| Si alguien está en París entonces no está en Moscú | C1:$\neg$P(x, París) $\vee$ $\neg$P(x, Moscú) |
| Juan está en París | C2: P(Juan, París) |
| ¿Juan está en Moscú? | C0:$\neg$P(Juan, Moscú) |

Puede verse fácilmente que no es posible deducir $\square$ de {C0,C1,C2}, por lo que la respuesta debería ser "No, Juan no está en Moscú"

- Si no se puede demostrar la conclusión, puede intentarse demostrar su negación

- Si no se puede demostrar ni la conclusión ni su negación, la respuesta debería ser "no hay suficiente información"

# Extracción de respuestas: Tipo B.

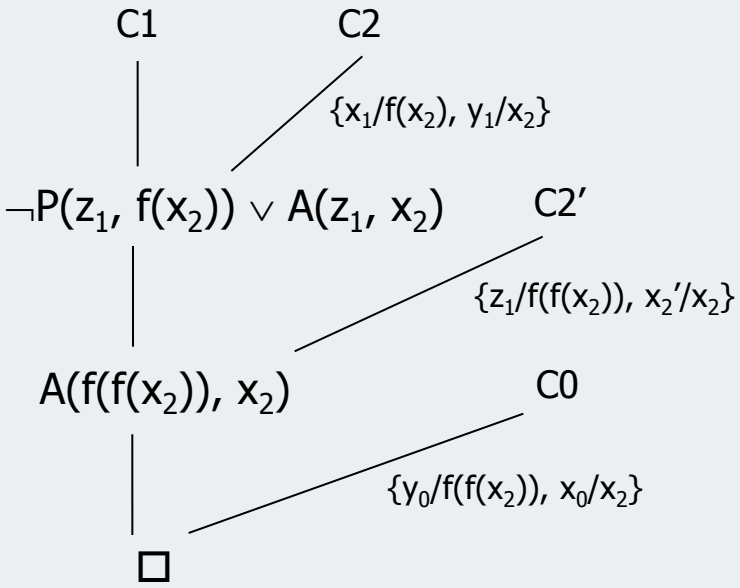


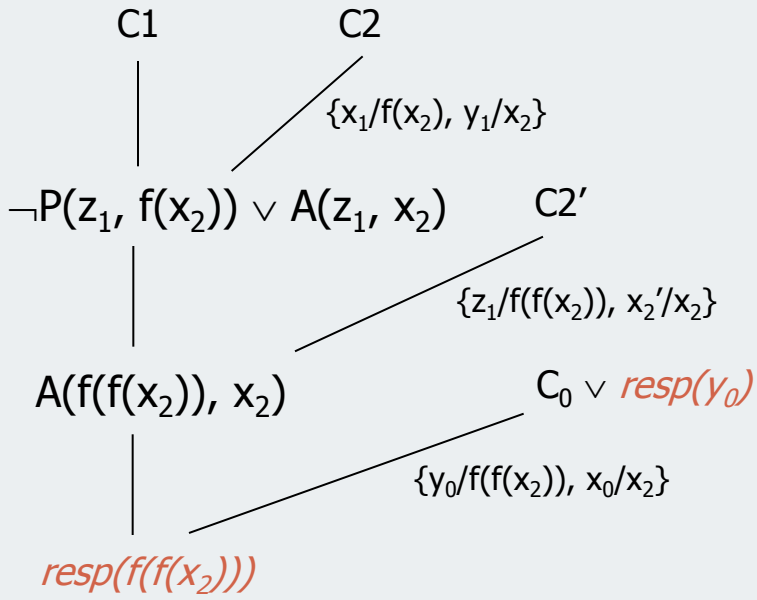
- Ejemplo:
  - Si *x* es padre de *y* y *z* es padre de *x* entonces *z* es abuelo de *y*
  - Todo el mundo tiene un padre
  - ¿quién es el abuelo de *x*?

C1: $\neg P(x, y) \vee \neg P(z, x) \vee A(z, y)$

C2. $P(f(x), x)$

C0: $\neg A(y, x)$



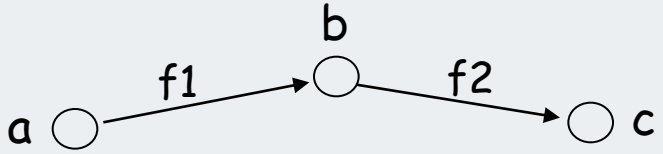

*cláusula respuesta*

# Extracción de respuestas: Tipo C.



- **En este tipo de preguntas la tarea es encontrar una secuencia de acciones con las que alcanzar un objetivo**
- **Idea:** Se asume que, en cada momento, cada objeto se encuentra en un cierto estado. Para alcanzar el objetivo hay que cambiar dicho estado al estado deseado. La demostración automática de teoremas se puede usar para encontrar las acciones con las que producir esos cambios

- Ejemplo:

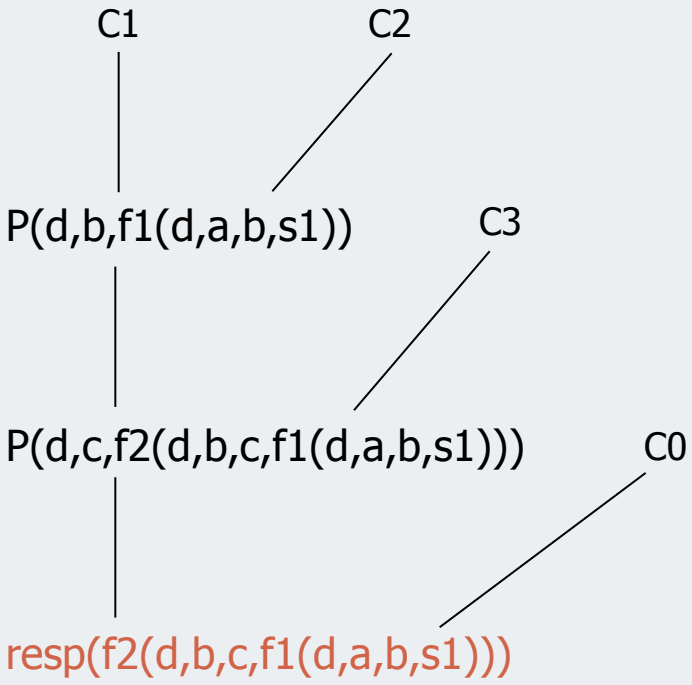



  - P(x,y,z): *x* está en *y* en el estado *z*
  - f1(x,a,b,z): estado al que pasa el sujeto *x*, en el estado *z*, al moverlo de *a* a *b* (acción necesaria para mover un sujeto de *a* a *b*)
  - f2(x,b,c,z): estado al que pasa el sujeto *x*, en el estado *z*, al moverlo de *b* a *c* (acción necesaria para mover un sujeto de *b* a *c*)

# Extracción de respuestas: Tipo C.



- ¿Cómo puede llegar *d* hasta *c*?
- El estado inicial del sujeto *d* es *s1* y se encuentra en *a* ➔ P(d,a,s1): C1
- $\forall z(P(d,a,z) \rightarrow P(d,b,f1(d,a,b,z)))$ ➔ $\neg P(d,a,z) \vee P(d,b,f1(d,a,b,z))$: C2
- $\forall z(P(d,b,z) \rightarrow P(d,c,f2(d,b,c,z)))$ ➔ $\neg P(d,b,z) \vee P(d,c,f2(d,b,c,z))$: C3




La respuesta se puede interpretar como la ejecución de dos acciones:

1) se aplica *f1* para llevar a *d* de *a* a *b*, y
2) se aplica *f2* para llevarle de *b* a *c*

# Extracción de respuestas: Tipo C.



- **El problema del mono y los plátanos:** Un mono quiere comer un plátano del racimo que cuelga del techo de una habitación. Aunque por sí mismo no lo alcanza, sí podría subirse a una silla que hay en la habitación para hacerlo.

## *Formalización*:

**Predicados**:

- *P(x,y,z,s):* en el estado *s*, el mono está en *x*, el plátano en *y* y la silla en *z*
- *R(s):* en el estado *s*, el mono puede alcanzar el plátano

**Funciones**:

- *camina(y,z,s):* estado alcanzado si el mono, partiendo del estado *s*, camina de *y* a *z*
- *arrastra(y,z,s):* estado alcanzado si el mono, partiendo del estado *s*, camina de *y* a *z* llevando con él la silla
- *escala(s):* estado alcanzado si el mono está en el estado *s* y se sube a la silla

# Extracción de respuestas: Tipo C.



- **Premisas** (o axiomas no lógicos):

1) Inicialmente el mono está en el estado *s1* en *a*, el plátano en *b* y la silla en *c*
2) El mono puede dirigirse hacia la silla (ir de *x* a *z*) en cualquier estado
3) Si el mono está al lado de la silla, que está en *x*, entonces puede arrastrarla hasta cualquier sitio *y*
4) Si el mono y la silla están ambos bajo los plátanos, el mono puede subirse a la silla y coger un plátano

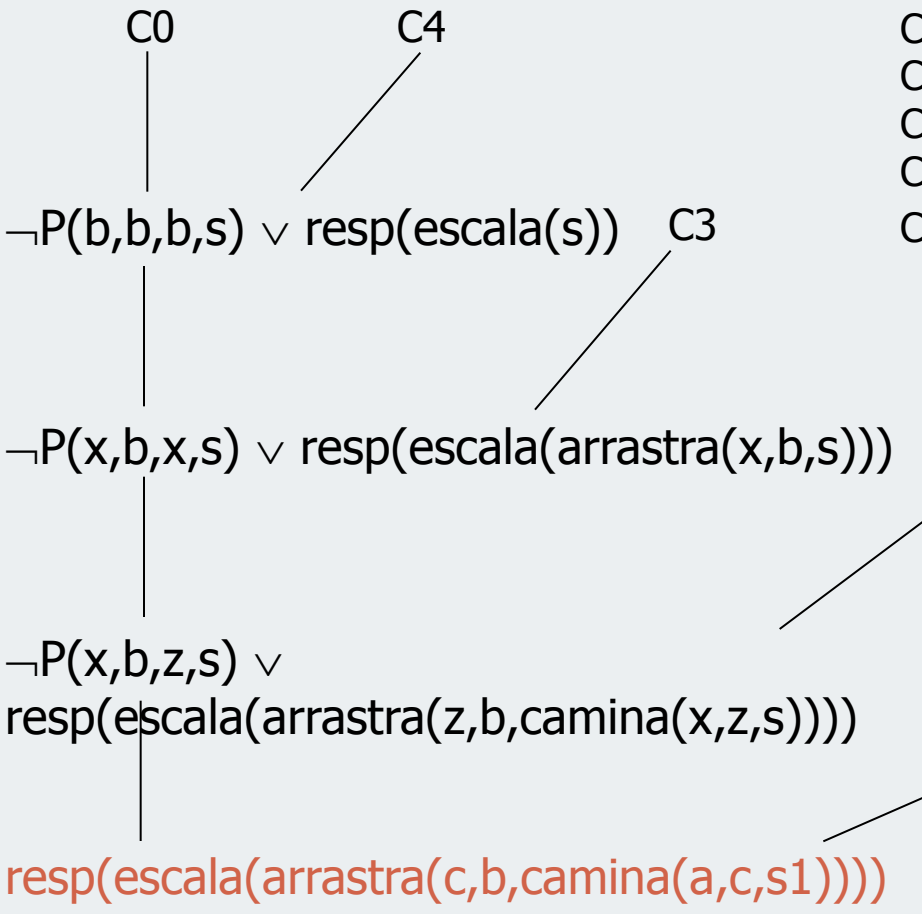
**Formalización de las premisas:**

1) $P(a,b,c,s1)$
2) $\forall x \forall y \forall z \forall s(P(x,y,z,s) \rightarrow P(z,y,z,camina(x,z,s)) \Rightarrow \neg P(x,y,z,s) \vee P(z,y,z,camina(x,z,s))$
3) $\forall x \forall y \forall z \forall s(P(x,y,x,s) \rightarrow P(y,y,y,arrastra(x,y,s)) \Rightarrow \neg P(x,y,x,s) \vee P(y,y,y,arrastra(x,y,s))$
4) $\forall s(P(b,b,b,s) \rightarrow R(escala(s)) \Rightarrow \neg P(b,b,b,s) \vee R(escala(s))$

**Pregunta**: ¿Qué tiene que hacer el mono para poder comer un plátano? ➔ $\neg R(s) \vee resp(s)$

# Extracción de respuestas: Tipo C.






C1: P(a,b,c,s1)
C2: ¬P(x,y,z,s) ∨ P(z,y,z,camina(x,z,s))
C3: ¬P(x,y,x,s) ∨ P(y,y,y,arrastra(x,y,s))
C4: ¬P(b,b,b,s) ∨ R(escala(s))
C0: ¬R(s) ∨ resp(s)

Respuesta:

1) El mono va de a a c
2) El mono va de c a b llevando consigo la silla
3) El mono se sube a la silla. Tras esta acción el mono ya puede alcanzar el plátano

# Extracción de respuestas: Tipo D.



- **En este tipo de preguntas la tarea es encontrar una secuencia de acciones, que bajo determinadas condiciones, pueda llevar al objetivo**

- **Idea:** Se asume que, en cada momento, cada objeto se encuentra en un cierto estado. Para alcanzar el objetivo hay que cambiar dicho estado al estado deseado.

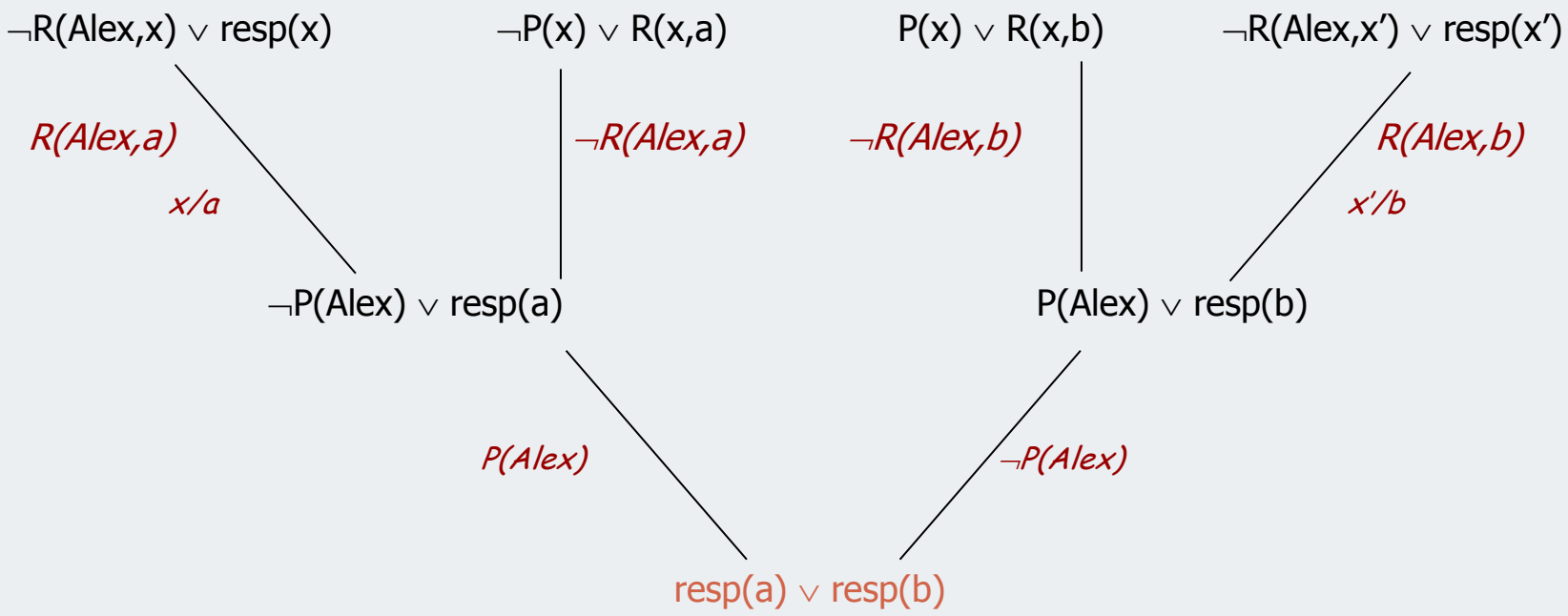
- La demostración automática de teoremas se puede usar para encontrar las acciones con las que producir esos cambios, pero la aplicación de esas acciones puede estar condicionada a la verificación de determinadas condiciones

- El árbol de resolución puede transformase en un árbol de decisión introduciendo un algoritmo de extracción de información

# Extracción de respuestas: Tipo D.



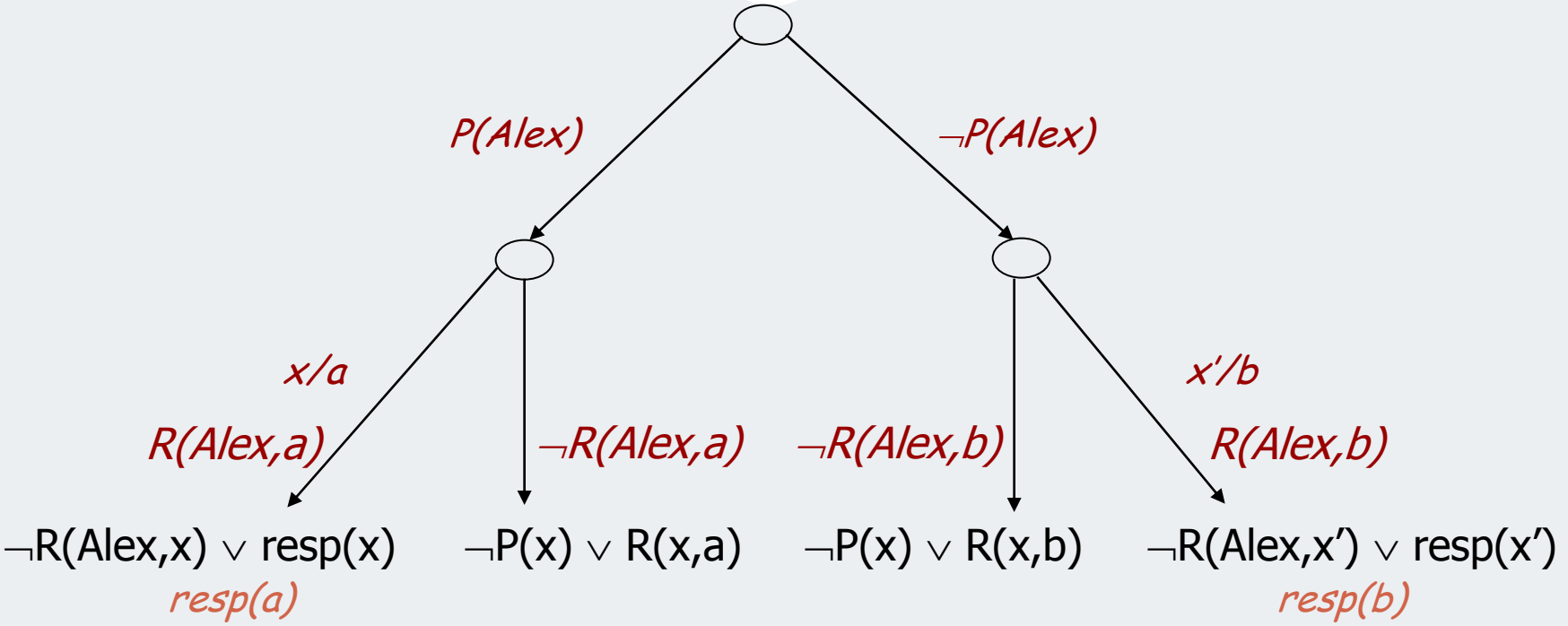
- Si alguien es menor de 5 años debe tomar la medicina a
  - $\forall x(P(x) \rightarrow R(x,a)) \rightarrow \neg P(x) \vee R(x,a)$: C1
- Si alguien no es menor de 5 años debe tomar la medicina b
  - $\forall x(\neg P(x) \rightarrow R(x,b)) \rightarrow P(x) \vee R(x,b)$: C2
- ¿Qué medicina debe tomar Alex?
  - $\exists x R(Alex,x) \rightarrow \neg R(Alex,x) \vee resp(x)$: C0




1) **Se etiquetan los arcos con los factores y el signo opuesto al de la cláusula de partida del arco**

# Extracción de respuestas: Tipo D.





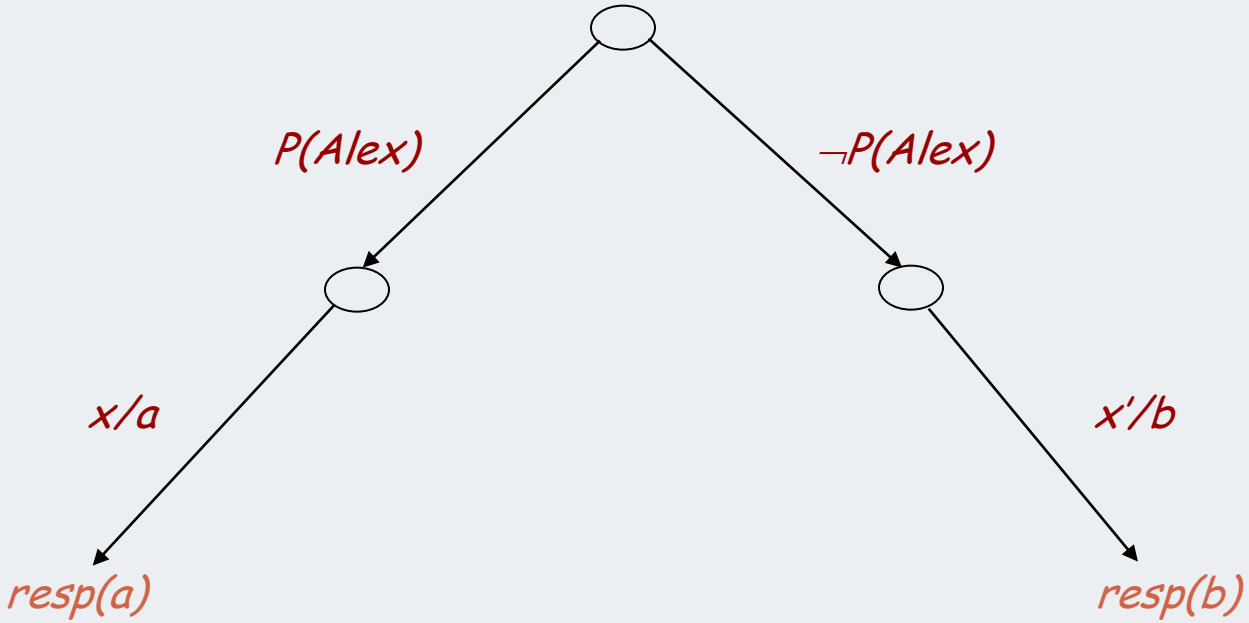

2) **Se invierte el árbol de resolución y se eliminan las cláusulas de los nodos no hoja**

3) **No se consideran las arcos y nodos que llevan a cláusulas que no incluyen el literal resp**

4) **De {P(Alex), R(Alex,a), ¬R(Alex,x) ∨ resp(x)} se deduce resp(a)**
**De {¬P(Alex), R(Alex,b), ¬R(Alex,x') ∨ resp(x')} se deduce resp(b)**

# Extracción de respuestas: Tipo D.






5) **Este árbol se puede considerar un árbol de decisión: si P(Alex) es verdad, entonces resp(a) es verdad, y si P(Alex) es falso, entonces resp(b) es verdad**

6) **Por tanto la respuesta es:**

*Si Alex es menor de 5 años entonces debe tomarse la medicina a, en otro caso debe tomarse la medicina b*

# Extracción de respuestas.



- **La resolución es completa para la extracción de respuestas:** si una pregunta tiene respuesta, entonces puede deducirse una cláusula respuesta por resolución

  - **Sea {C1, C2, …, Ck} un conjunto de cláusulas que representa a un conjunto de hechos. Si se formula una pregunta como: "Encuentra valores para x1,…,xn tal que la afirmación P(x1,…,xn) sea cierta", la pregunta tiene respuesta sii [C1, C2, …, Ck] |— ∃x1… ∃xnP(x1,…,xn)**
  - **Sea C2 la cláusula: ¬P(x1,…,xn) ∨ resp(x1,…,xn)**

  **Teorema: Sea C = C1 ∪ C2. La pregunta tiene respuesta sii existe una deducción de una cláusula respuesta a partir de C**

- **La resolución no sólo nos dice si la pregunta tiene respuesta sino también cuál es esa respuesta**

# Programación lógica. Programas.



- Una cláusula de Horn que no tiene literales negados se denomina **hecho**

- Una cláusula de Horn que tiene un literal afirmado y el resto negados se denomina **regla**. El literal afirmado se llama **cabeza** y la secuencia de literales negados se llama **cuerpo**

| | | | |
|---|---|---|---|
| $A \vee \neg B1 \vee \neg B2$ | ➔ $B1 \wedge B2 \rightarrow A$ | ➔ $A \leftarrow B1,B2.$ | *A es cierto si B1 y B2* |
| $A$ | ➔ $A$ | ➔ A. | *A es cierto sin condiciones* |
| | *equivalencias sintácticas* | *sintáxis prolog* | |

- Un conjunto de reglas con la misma cabeza es un **procedimiento**, cuyo nombre es el del símbolo de predicado correspondiente a la cabeza

- Un conjunto de definiciones de procedimientos es un **programa** lógico

# Programación lógica. Consultas.



- Una cláusula de Horn que no tiene literal afirmado se denomina ***objetivo***
- Un objetivo se escribe como si fuera un cuerpo y se considera una pregunta

$\neg B1 \vee \neg B2$ ➔ $B1 \wedge B2 \rightarrow$ ➔ $\leftarrow B1,B2.$ *¿Es cierto B1 y B2?*

- La deducción cuya corrección se debe comprobar es la que tiene como premisas al programa y como conclusión al objetivo.
- La **ejecución** de un programa lógico con un objetivo dado consiste en determinar si el objetivo es deducible del programa, y caso de que lo sea, los valores de las variables del objetivo que dan una respuesta al mismo.
- Para la ejecución se aplica resolución SLD con el objetivo como cláusula objetivo inicial.
- La búsqueda de la refutación se hace en profundidad con vuelta atrás (*backtraking*).

# Ejemplo.



1. padre(A,B)
2. madre(B,C)
3. abuelo(x,z)← progenitor(x,y), progenitor(y,z)
4. progenitor(x,y) ← padre(x,y)
5. progenitor(x,y) ← madre(x,y)

**¿Quién es abuelo de C?:** abuelo(x,C)

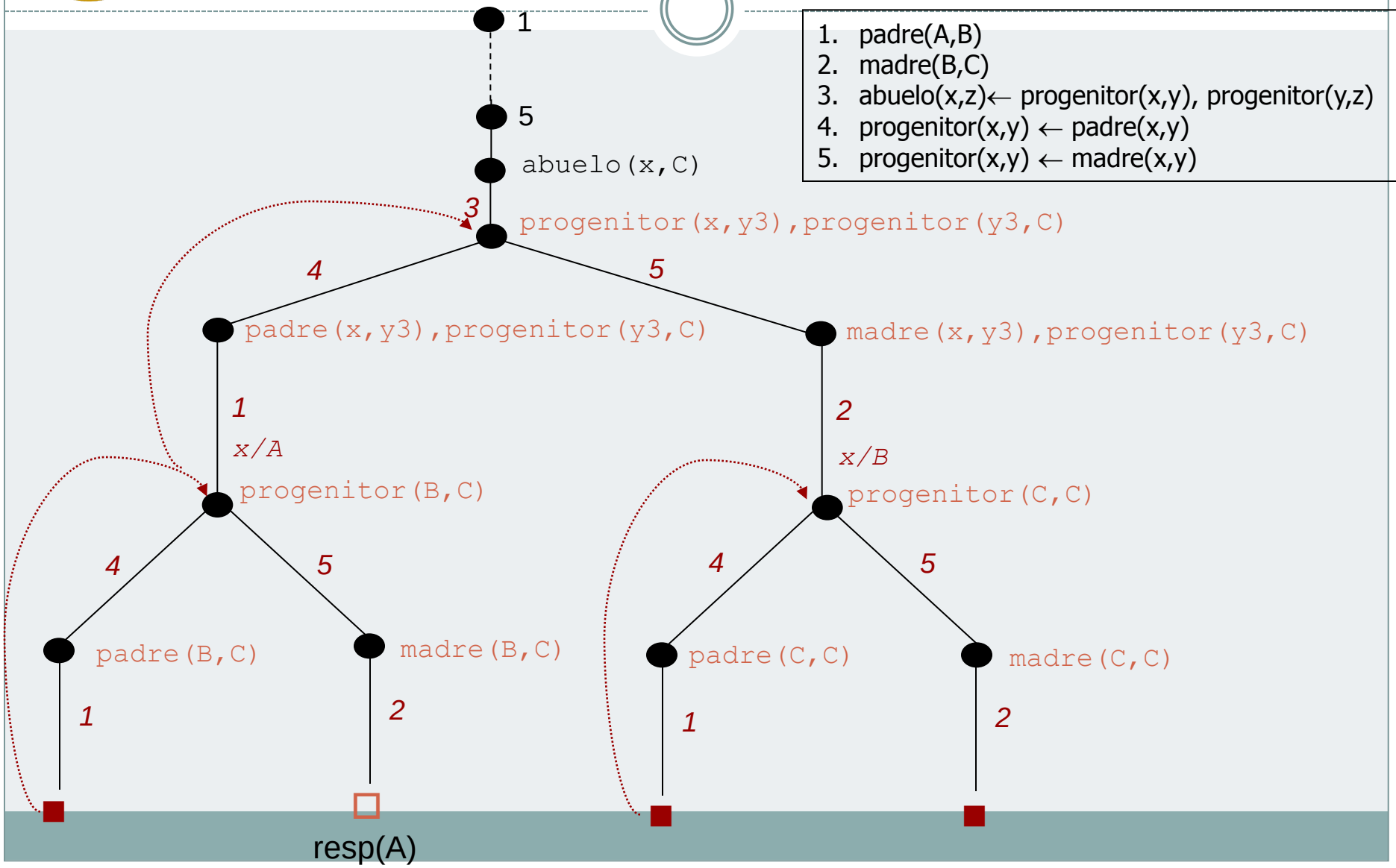
- abuelo(x,C),resp(x)
  [3,x3/x,z3/C]
- progenitor(x,y3),progenitor(y3,C),resp(x)
  [4,x4/x,y4/y3]
- padre(x,y3),progenitor(y3,C),resp(x)
  [1,x/A,y3/B]
- progenitor(B,C),resp(A)
  [4,x4'/B,y4'/C]
- padre(B,C),resp(A)
  **<fallo>**
  [5,x5/B,y5/C]
- madre(B,C),resp(A)
  [2]
- **resp(A)**

**¿Quién es nieto de A?:** abuelo(A,x)

- abuelo(A,x),resp(x)
  [3,x3/A,z3/x]
- progenitor(A,y3),progenitor(y3,x),resp(x)
  [4,x4/A,y4/y3]
- padre(A,y3),progenitor(y3,x),resp(x)
  [1,y3/B]
- progenitor(B,x),resp(x)
  [4,x4'/B,y4'/x]
- padre(B,x),resp(x)
  **<fallo>**
  [5,x5/B,y5/x]
- madre(B,x),resp(x)
  [2,x/C]
- **resp(C)**

# Ejemplo.

1. padre(A,B)
2. madre(B,C)
3. abuelo(x,z)← progenitor(x,y), progenitor(y,z)
4. progenitor(x,y) ← padre(x,y)
5. progenitor(x,y) ← madre(x,y)




# Inversión de una lista en Pascal

**Pascal**

```
program reverse(output, intfile, outfile)
type      ptr = ^item;
          item = record
          val : integer;
          next : ptr;
          end;
     var x : integer;
          p, list : ptr;
          intfile, newfile : file of integer;
begin
reset(intfile); rewrite(newfile);      (* read in a sequence *)
list := nil;                           (* initialize the list *)
read(intfile, x);
while x <> 0 do
       begin
       new(p);
       p^ . val := x;                  (* record the integer *)
       p^ . next := list;              (* and link into *)
       list := p;                      (* the list *)
       read(intfile, x)
       end;
p := list;                             (* output the sequence in reverse *)
while p <> nil do
       begin
       write(newfile, p^ . val);
       p := p^ . next                  (* advance down the chain *)
       end
end.
```

# Ejemplo.



Inversión de una lista en Prolog

```
reverse([ ], [ ]).
reverse([X|R], S) :- reverse(R,L), conc(L,[X],S).
```

Prolog sencillo y fácil de instalar: www.swi-prolog.org